The Role of Parts-Of-Speech in Java Program Identifiers

Meilani Williams, Emily Hill, Lori Pollock, and Vijay Shanker
Computer & Information Science
{mwilliam, hill, pollock, vijay}@cis.udel.edu

ABSTRACT

In today's largest and complex software, a piece of code will need to be read and understood by many software developers. To communicate their thought processes in code, developers use meaningful identifier names. Thus, identifiers capture what a developer intends to accomplish with a portion of code. We have found that natural language clues in identifiers can improve automated software tools to increase developer program comprehension and facilitate software maintenance tasks. In this paper, we present a study of word usage in Java program identifiers. We present our case study methodology, examples of word usage, observations from the study, and how these observations can improve automated software tools. This research study was sponsored by the Computer Research Association for Women Distributed Mentor Program (CRA-W DMP).

1. INTRODUCTION

Today's large and complex code makes software maintenance tasks difficult. It takes developers a considerable amount of time to understand the software system well enough to make correct maintenance changes: throughout the life cycle of an application, as much as 60-90% of resources are devoted to modifying the application to meet new requirements and fix faults [2]. Software maintenance is more expensive when code is not easy for developers to understand. Therefore, in order to decrease software maintenance time and expense, software tools must be able to help developers quickly comprehend and effectively modify code.

To reduce the cost of software maintenance, previous work [5,6] has demonstrated that natural language clues in program identifiers can be used to improve software tools. Because developers use meaningful identifier names to communicate thought processes in code, identifiers capture what a developer intends to accomplish with a portion of code [4]. Thus, software tool developers can leverage these natural clues found in meaningful identifiers to help developers better understand code.

In program identifiers, developers may use English words in a specific way. For example, consider the preposition 'to' in the method name `toString`. The 'to' actually implies a conversion action, such as 'convert this object to a string'. A software search tool can leverage this information to successfully return relevant methods like 'to String' for a query such as 'convert string'.

Thus by studying word usage in Java program identifies we can find natural language clues like 'to' that can be used to improve software tools. Based on our observation of how the preposition 'to' was

used, we decided to research parts of speech (POS) with a focus on preposition word usage and the potential effect on software tools. Several researchers have studied the effect of POS on Java program identifiers, but they focused on the relationship of noun and verb usage in code [4,6]. In this paper, we investigate how programmers use prepositions in Java program identifiers. Our study covers the specific use of prepositions in field, method, and class declarations. We used these observations to show how Java program identifiers can improve software tools to increase developer program comprehension and facilitate software maintenance tasks.

2. RELATED WORK

In spite of the fact that identifier names do not affect program execution, existing work has demonstrated that developers do select meaningful identifier names [1,4]. Liblit et al. discovered that method names may be defined using verbs in different moods, and that these moods can provide insights into a method's functionality [4]. For example, imperative verbs such as `add, addAll, remove,` and `set` tend to indicate an action being taken. In contrast, indicative verbs such as `contains` or `equals` imply a function returns a factual, boolean assertion. Finally, methods that simply return objects tend to have names composed of noun phrases: `capacity, clone, lastIndexOf, size`.

As a result, it is possible to use the natural language clues in identifiers to improve software tools. For instance, Shepherd et al. used noun and verb usage in identifiers to develop a concern location tool [6]. This same group later used natural language clues such as antonym verb usage to improve an automated aspect mining tool [5]. Both works influenced our research in the use of prepositions as natural language clues in Java identifiers.

3. METHODOLOGY

We developed an automated software analysis tool to extract preposition word usage samples. For each field name we extracted the field's declaration and two succeeding lines of code. For method declarations we extracted the declaration along with five succeeding lines of code. For classes we extracted the class declaration plus ten lines of succeeding code. We also extracted the leading comment for any field, method, or class declaration for each sample.

Figure 1 shows the 15 subject applications we extracted samples from. To do this we tokenized all field, method, and class identifier declarations by splitting on punctuation and camel case (for example, the identifier `ASTVisitor` would be split into the terms '`AST`' and '`Visitor`'). We then used the UNIX `spell` dictionary to remove tokens that are not words, thus eliminating abbreviations and other non-words. To determine which words were prepositions, we used the CLAWS4 part of speech tagger [3]. Finally, we extracted samples of prepositions from the subject programs. We analyzed prepositions that occurred at least 250 or more times in the subject applications. We noted the similarities and differences of how prepositions were used in identifiers versus English text which determined whether the preposition could be used to improve software tools.

Program	Program Description	# Lines of Code	# Methods of Code
Dguitar	a viewer for Guitar Pro tablature files	13,322	1,211
Drawswf	animated SWF drawing application	27,674	2,747
Freemind	mind map manager	70,341	6,110
Gantt	client application to plan projects using Gantt Charts	43,245	4,941
Ireport	report generation and editing application	74,392	5,672
J2SE1.5	Java API Implementation	1,509,015	115,790
Jajuk	music player and organizer	30,847	2,137
JavaHMO	access and display media	23,797	1,532
Jbidwatcher10pre6	online auction sniping application	22,997	1,918
Jftp	Java FTP implementation	34,426	2,379
JhotDraw	Java drawing application	39,179	4,267
Jrobin	chart and graph generator	19,469	1,913
MegaMek	Java implementation of BattleTech	147,404	9,256
PlanetaMessenger	An Instant Messenger (IM)	11,125	1,142
Prefuse	graph visualization toolkit	40,956	5,023

Figure 1. Subject Applications

4. RESULTS

to	• `SequenceToMidiTrackEvents()` • `applyStylesTo(Element elem)` • `fireOnAddUserToContactList(String strUserId)` • `ResourceAssignment myAssignmentToTask;`
as	• `addASModel(ASModel abstractSchema)` • `WildcardType asWildcardType()` • `boardSaveAs()` • `addASModel(ASModel abstractSchema)` • `JmenuItem fileBoardSaveAsImage`

Table 1. Preposition Usage Samples for 'to' and 'as'

In our study, we found that four prepositions had specific meanings different than English text. The prepositions “to” and “as” imply nonexistent verbs within programs. These words were used for conversions; for example the `toString()`method converts an object to a string. The word “as” also indicates that a conversion has occurred within a program or a specific object, for example, the `getGradientAsSVG()`method, which converts a gradient to an SVG object. Table 1 shows more examples of ‘to’ and ‘as’ indicating conversions. Software tools such as search tools can leverage this information to provide improved search results. When a developer is searching through the code for methods dealing with conversions the search tool would be able to extract relevant methods that include “to” and “as”.

next	• `boolean nextPage()` • `class NextVisualPositionAction` • `class NextWordAction` • `boolean hasNext()`
after	• `symbol symbol_after_dot()` • `String ALLOW_DTD_EVENTS_AFTER_ENDDTD_FEATURE = "allow-dtd-events-after-endDTD"` • `boolean isAfterLast()` • `void reportInterruptAfterWait(int interruptMode)`

Table 2. Preposition Usage Samples for 'next' and 'after'

We noted similar patterns with other prepositions. 'Next' and 'after' indicate time or position. Additional samples of how these words are used to represent time and position can be seen in Table 2. We note that 'next' is an adjective, but within Java program identifiers it can act as a preposition (for example the `getNextTrackPanel()` indicates that this method retrieves the following track panel; it is acting on that object). Because next indicates the order or sequence of events that must be performed in a certain portion of code, it can be used as a beacon to help automatically summarize algorithms in code.

The preposition 'after' acts in the same manner as 'next', by denoting an object's placement or a reference to time (for example, `ScheduleAfterActivity`, which indicates when the following task should be scheduled). Further, this natural language clue can give insight into the content of the class, `ScheduleAfterActivity` (meaning that the methods within this class will be focused on how and when to schedule certain tasks). By leveraging how 'after' is used within identifiers software tools can help increase developer program comprehension. With an overall summary of the algorithm the developer spends less time trying to understand what is being implemented.

We analyzed other prepositions that were used similarly in programs as in English text. At present, we were unable to determine how these prepositions could be leveraged to improve software tools.

- at: `public int getLevelAt(int offset), public static int GRADE_CHECK_MODIFIED_AT_COMMIT = 2;, public void addProviderAtFront`
- from: `public static final GP4Slide FROM_BELOW = new GP4Slide(- 1), private NumberFormat.Field getFieldFrom(int index, int direction), private Class fromClass;`
- in: `public void storeInRegister( int registerNumber ), public final boolean inSamePackage(ClassDefinition c), boolean inProgress;`
- of: `private int lengthOfTask , public static String valueOf(boolean b), private boolean atEndOfData;`
- on: `private class GP3EffectsOnBeat, public void centerOnHex(Coords c), public class CopyOnWriteArrayList`
- into: `public void drawInto(Graphics g), public static final int`

```
GET_URL_MODE_LOAD_VARS_INTO_LEVEL   = 3;
```

- up: `final public long getSysUpTime(), public static final String pageUpAction = "page-up", protected void setUp()`
- with: public class ByteBufferWithInfo, public Hashtable getElementsWithIDs(), public final static short SQL_SUCCESS_WITH_INFO = 1;
- within: `private void checkWithinBounds(List<Type> tvars, List<Type> arguments, Warner warn), protected JFrame m_within, private boolean moveWithinSelectedRange`
- without: `protected void writeLongWithoutAlign(int x), public static Class loadClassWithout(ClassLoader loader,  String className), public static final boolean WITHOUT_CONTEXT = false;`

Take for example the preposition "of". For the method `velocityOf(GP4Dynamic dynamic)`, "of" is used the same in the Java identifier as in English. Another example of similar use is "at" in a field declaration `atInsertRow`. , which refers to row placement giving insight into an order of when an event will execute. Thus, the prepositions that are used similarly to English give no natural language clues beyond their English definitions.

5. CONCLUSIONS & FUTURE WORK

By studying the use of prepositions in Java program identifiers, we gained valuable insights into natural language clues that can be used to improve software tools such as searching and automatic algorithm summarization. In the future, we plan to analyze more prepositions and analyze word usage samples from different subject programs, integrate our observations into existing software tools, and evaluate our rules' effectiveness. Further analysis will also enable us to better evaluate how such natural language clues can improve software tools.

6. REFERENCES

[1] B. Caprile and P. Tonella. Nomen est omen: Analyzing the language of function identifiers. IN *WCRE' 99: Proceedings of the Sixth Working Conference on Reverse Engineering,* page 112, 1999.

[2]L. Erlikh. Leveraging legacy system dollars for e-business. *IT Professional*, 2(3):17-23, 2000.

[3] R. Garside and N. Smith: A hybrid grammatical tagger: CLAWS4. In Garside, R., Leech, G., and McEnery, A. (eds.) *Corpus Annotation: Linguistic Information from Computer Text Corpora,* 1997.

[4] Ben Liblit, Andrew Begel, and Eve Sweeser. Cognitive perspectives on the role of naming in computer programs. In *Proceedings of the 18th Annual Psychology of Programming Workshop,* 2006.

[5] David Shepherd, Lori Pollock, and K. Vijay-Shanker, "*Case Study: Supplementing Program Analysis with Natural Language Analysis to Improve a Reverse Engineering Task*", Workshop on Program Analysis for Software Tools and Engineering (PASTE 2007), June 2007.

[6] David Shepherd , Zachary P. Fry, Emily Hill, Lori Pollock, and K. Vijay-Shanker. Using natural language program analysis to locate and understand action-oriented concerns. In *AOSD '07: Proceedings of the 6th international conference on Aspect-oriented software development,* 2007.